1. INTRODUCTION

The External Interface Specification (EIS) for the CHESS system provides detailed information about the protocol used to communicate with CHESS, the format and content of messages which are exchanged between CHESS users and the CHESS computer system, and the format and meaning of the fields within each message, including allowable data values and their meanings where applicable.

We recommend that system developers within user organisations and third party developers implement the full set of messages described in this document which are appropriate to their sector of the industry.

This specification is **not** intended as a guide to the functions performed by CHESS, nor to provide a management summary of the background to the project. These and other aspects of the project are described in other CHESS publications which are available from ASX and include:

- the CHESS Overview;
- Industry procedure guidelines.

The External Interface Specification is a controlled document. All updates are provided through ASXOnline as and when they take effect.

A print version of the EIS is also provided through ASXOnline.

2. TECHNICAL ARCHITECTURE & KEY COMPONENTS

2.1 Technical Architecture of the CHESS System

The technical architecture of CHESS encompasses the key physical aspects of the processing sites, computer systems and associated communications network. CHESS operates as a computer-to-computer system with exchange of electronic information between the clearing house computer systems and CHESS users' computer systems.

The technical architecture of CHESS is based on integration with existing ASX processing facilities which comprise:

- two processing sites located 10 km apart;
- multiple computer processors and disk storage media in both sites.

The CHESS Network is illustrated in Figure 2.1 below.

CHESS allows the alternative processing site to act as a warm standby (that is, able to resume processing within four hours of a failure).

The alternative processing site is also used for ongoing user testing of new releases of CHESS software and for communications testing for new users.

CHESS users connect to CHESS via either the Telstra Austpac network or the Optus eFinity network. All CHESS users are encouraged to migrate to the Optus eFinity network.

CHESS Network

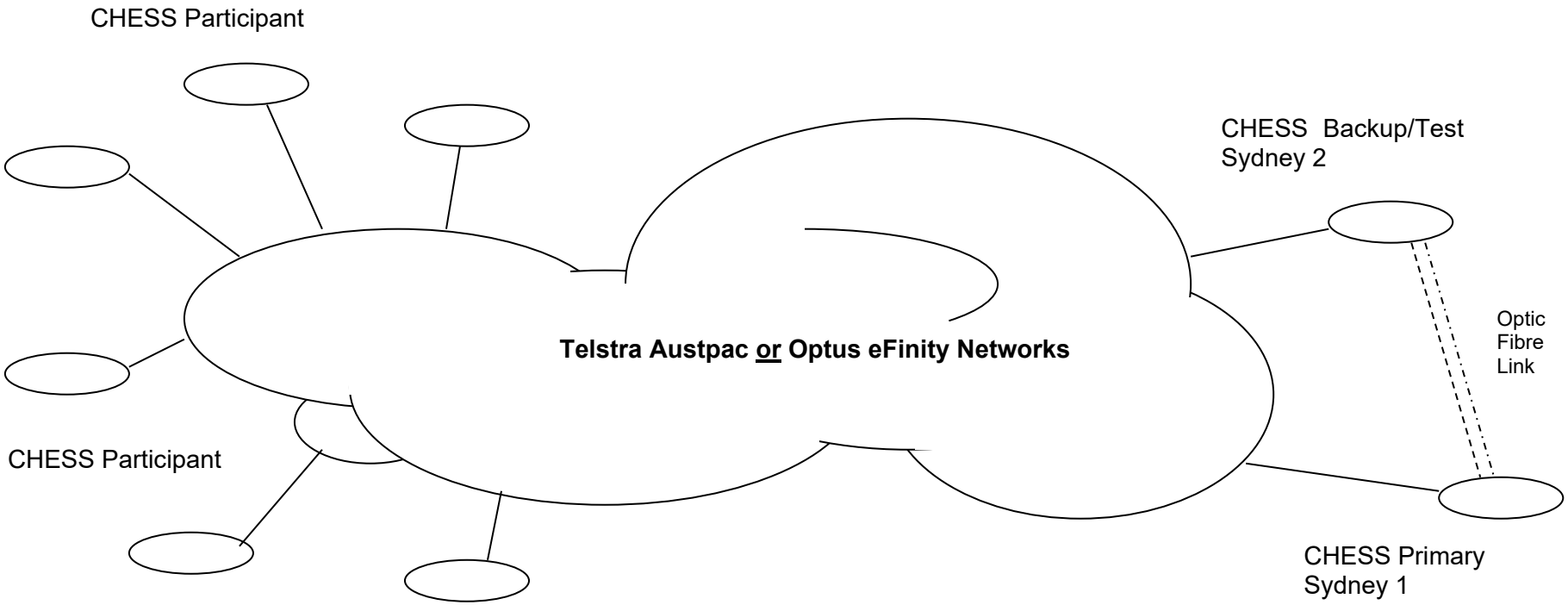


Figure 2.1

2.2 Key Components of the External Interface

The external interface to CHESS consists of a number of discrete components:

- the **message structure** describes the way in which message fields are identified within a message (Section 3).
- the **message types** describe the particular message fields and their sequence in each type of message (Section 4).
- the **data representation** describes the standard adopted for message fields in CHESS (Section 7).
- the **network definition** describes the alternatives for establishing a physical connection to CHESS together with the communications protocols required to transmit information to or receive information from CHESS (Section 10).
- the **communications security** includes mandatory and optional features that are authorised for use with CHESS (Section 11).

Each of these components is described in more detail in the appropriate section of this specification.

2.3 Standards

In conjunction with the preparation of this specification, a review of communications standards in the finance industry, including SWIFT, ISO 7775 and AS2805/ISO 8583 was undertaken. **Although elements of these existing standards have been used in the design of the CHESS interface, the decision has been taken not to adopt any particular standard *in toto*.**

2.4 Nomenclature

The nomenclature used in this specification conforms to that commonly used in communications standards.

3. CHESS MESSAGE STRUCTURE

3.1. Principles

The CHESS data message structure is designed to satisfy the following objectives:

- minimise the amount of information sent;
- enable the receiver to validate, simply and easily, the format of the message;
- minimise the extent of system changes if the format of a message type changes.

Each type of message allows a business function to be carried out. Any particular message type may consist of mandatory and optional fields. This improves efficiency in terms of system/communications capacity and cost as it minimises the amount of information sent in any message.

The receiver of a CHESS message is able to inspect the message to validate its correctness and integrity. The message structure is such that changes can be easily made to a particular message type, and the changed message type can be incorporated into all relevant communicating systems with minimal programming and system design changes. The message structure adopted for CHESS minimises the possibility of older versions of message types being used accidentally.

The CHESS message format is commonly called a **bit map structure**. It is used, for example, in the AS2805 and ISO 8583 standards.

3.2. Message Structure Implementation

The CHESS Data message is based on the bit map structure. The structure of each message is defined by the bit map for that particular message type as given in Section 4. Definition of the overall structure of messages is given in Section 10.5.

The bit map is a 64 bit field which occurs after the message type in each message. The presence or absence of a field in the message is indicated by a bit in the bit map field.

The first bit of the bit map indicates the presence or absence of a bit map extension, and the next 63 bits refer to specific CHESS fields. If a particular message defines fields subsequent to the 63 referred to by the bit map, then a bit map extension is used.

Mandatory fields, and optional fields (if present) in a particular instance of a message type, have the appropriate bit set. Unused fields have the bit clear in the bit map.

If a bit map extension is defined for a particular message but only has optional fields and a user chooses to omit all of those fields, the bit map extension should not be included. In this case, the 'continuation bit' in the first bitmap should be clear and the second bitmap should not be present, with data fields starting immediately following the first bitmap. If a third bitmap were to be defined, then a similar relationship would exist between the second and third bitmaps. If data were present in a third bitmap, then the second bitmap is required, even if only to indicate the presence of the third bitmap. If a fourth bitmap were to be defined, then a similar relationship would exist between the third and fourth bitmaps. If data were present in a fourth bitmap, then the third bitmap is required, even if only to indicate the presence of the fourth bitmap.

Validation of the bit mapped messages is comparatively simple. The tabular structure of the format implies a simple implementation for validation and inspection. The bit map supplied in a particular message is checked against the general bit map for that message type for correctness. The message fields can be checked for format errors based on the general table.

Modifications can be made by incorporating new fields or message types in the general bit map. Existing message types can have their component fields easily modified or deleted.

The bit map message structure satisfies the criteria of minimising overheads, being simple to validate and adaptable to change.

When developing code for the implementation of the bit mapped messages, users should always endeavour to refer to the message's bit map to detect the presence or absence of a field rather than relying on a pre-determined fixed message structure interpreted from the specification. This will lead to easier incorporation and tolerance of changes to message formats.

Where optional fields are not present in messages sent by users to CHESS, CHESS will default these fields to either numeric zero, or character blank depending on their data type.

3.3. Examples

An example of the bit map usage for a Holding Balance message is shown in Figure 3.1 below.

Section 4 specifies which bit position corresponds to a particular field in a given message.

BIT	VALUE	MESSAGE FIELDS	FORMAT	TYPE `522'
1	0	Secondary Bit Map	64 bits	clear
2	1	Security Code	12 character	mandatory
3..15	0			
16	1	HIN	10 numeric	mandatory
17..20	0			
21	1	Processing Timestamp	22 character	mandatory
22..52	0			
53	1	Total Unit Balance	11 numeric	mandatory
54..61	0			
62	1	Origin Transaction ID	16 character	mandatory
63..64	0			

FIGURE 3.1: BIT MAP STRUCTURE FOR HOLDING BALANCE

An example of the Holding Balance message using the bit map approach is shown in Figure 3.2 below.

Field Name	Value
Message Type	522
Bit Map	X'4001080000000804'
Security Code	AU000000WBC1
HIN	1234567890
Processing Timestamp	1993041919930419180430
Total Unit Balance	00000015000
Origin Transaction ID	1099042042000100

FIGURE 3.2: EXAMPLE OF A BIT MAPPED MESSAGE FOR HOLDING BALANCE

An example of the Tax File Number / Australian Business Number Advice message is shown in Figure 3.3 below. This message illustrates the use of a bit map extension, and also contains optional fields (refer to Message Type 533 in Section 4). Note that the bits for optional fields which are not present are set off (zero).

Field Name	Value
Message Type	533
Bit Map	X'C001000000010000'
Second Bit Map	X'0000000010000000'
Security Code	AU000000WBC1
HIN	1234567890
Transaction Id	1099042042042100
Tax File Number / ABN 1	01234567898C

FIGURE 3.3: EXAMPLE OF THE TAX FILE NUMBER / AUSTRALIAN BUSINESS NUMBER ADVICE MESSAGE

The definitions for the different data formats are given in Section 7.

The packaging of this section of data within the complete CHESS application protocol transaction is specified in Section 10.

3.4. Bit Map Technical Representation

The examples above showed the bitmaps as a series of 'hexadecimal' characters. When interpreting these from **documentation**, the following convention applies:

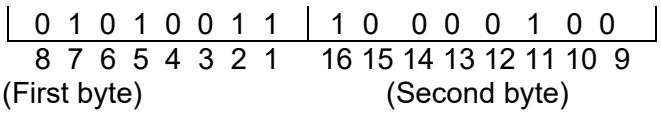
> Write the selected bits as a sequence of 0s and 1s from left to right for the full length of the bitmap(s) and separate these into groups of 4 digits. For example, considering the first 16 bits of a hypothetical bitmap which has bits 1, 2, 5, 7, 11, and 16 set, these are written (starting from bit number 1 on the left) as
>
> 1100 1010 0010 0001
>
> Next treat each group of 4 bits as if it were a hexadecimal number and write these as follows:
>
> CA 21

This is how the above examples were constructed **for documentation purposes only**.

When considering the **physical implementation** as observed when analysing memory or disc file structures, the results will depend on the specific hardware platform being used.

Hardware Architecture Type 1

On some hardware, for example an **IBM PC**, bits in a byte are stored **least significant bit to the right** and thus if a set of 16 bits is defined, the above example would be stored as follows (in what appears to be reversed order per byte) :



Hence examination of this example from a memory or disc file dump would appear as :

53 84 (hexadecimal)

even though the bitmap was documented as :

CA 21.

This is to be expected and is quite correct - there is no need to change the order.

Hardware Architecture Type 2

On a hardware platform which implemented the storage of bits within bytes with **least significant bit leftmost**, the **physical storage would appear as matching the documented layout**.